Modeling Project Evaluation Criteria

The mark for the Modelica & Bond Graph course is based, for 2/3, on a modeling project that is carried out over the last sessions (and which requires additional work outside class). Here are the evaluation criteria for this project.

1 Background for the project

The modeling project aims to put into practice, on a system of reasonable complexity, the different learning objectives of the course which I remind here:

- **To use the Modelica language and environment** to model and simulate dynamical systems (usually belonging to several physical domains). To be able to reuse standard Modelica components and to create custom ones.
- **To structure a complex model** into reusable parts. The architecture must be understandable and correspond to that of the system being modeled.
- **To work in a team** on the same complex model. The model is developed collaboratively and may be easily reused by third parties.

2 Realism of the Model

Given the 3 course objectives mentioned above, the goal of the project is *not to create the most accurate model* of the modeled system. Rather, it is expected that the model you produce will be, in order of priority:

1. **Functional**: no compilation errors, the model simulates “something”, not necessarily very realistic
2. **Credible**: the results are reasonable (correct orders of magnitude for the different simulated quantities), but the physics of the components is absent or very simplified.
3. **Realistic**: the components are modeled in a physically realistic way, even very detailed for some of them.

Level 1 (functional) is important and necessary, but not enough to validate the project. Level 2 is the minimal target to validate and level 3 unlocks the very best grades.

3 Objectives related to handling complexity

Remark: some objectives that I classified under “Structuring” also correspond to "Reusability by third parties", and conversely, because structuring and reusability are converging goals!

3.1 Model structuring

- Reasonable architecture: the Modelica model is structured into physical components that indeed correspond to subsystems of the modeled system.

- Those components are designed *as if* to be reused in different assemblies
  - Stand-alone tests: one or a few of the most complex components are tested autonomously in dedicated test models.
  - Parameterization: all components have appropriate parameters and variables (they clearly correspond to parameters of the modeled object). Their name is judicious, and the parameters have a default value if appropriate.
- Use of inheritance ("extends"), if necessary, to avoid code repetition (DRY principle).
- Packaging: the set of classes which make up the model is structured in one package and sub-packages according to a reasonable tree structure.
  - Custom connectors and partial classes, if any, are in a sub-package "Interfaces" like in the Modelica Standard Library.

3.2 Teamwork: Collaborative development process

Objective: The model is developed collaboratively using version control. Details:

- The model is delivered in a git repository on gitlab-student.centralesupelec.fr
- The history of the repository ('git log') reflects the evolution of the code during the project
- The history of the repository reflects the collaboration between the different people (each one created its own commits)
- Each developer has one (or a few) specific role within the project: architecture, expertise on a component, integration testing…

3.3 Teamwork: Reusability by third parties

Objective: a potential third party can quickly and autonomously take over the model and the entire set of components (just to run simulations but also modify the code).

- README instructions: For future users or developers of the model, the root folder of the project contains a README file. This text file:
  - presents the project and the names of the developers
  - quickly describes the structure of the code (package layout)
  - gives the entry point for a user: which model to open to make a simulation
  - is optionally formatted in Markdown (i.e. README.md file).
- Cleanliness: the final version of the git repository does not contain unnecessary files anymore (old classes, temporary files…).
- Ease of simulation: models are easy to simulate because the default settings (in particular the StopTime) are well set.
- Code documentation:
  - The models and their attributes (variables, parameters, ports…) are documented by a descriptive string.
  - Variables and parameters have an appropriate physical type (typically from Modelica.SIunits), with the use of displayUnit if necessary.